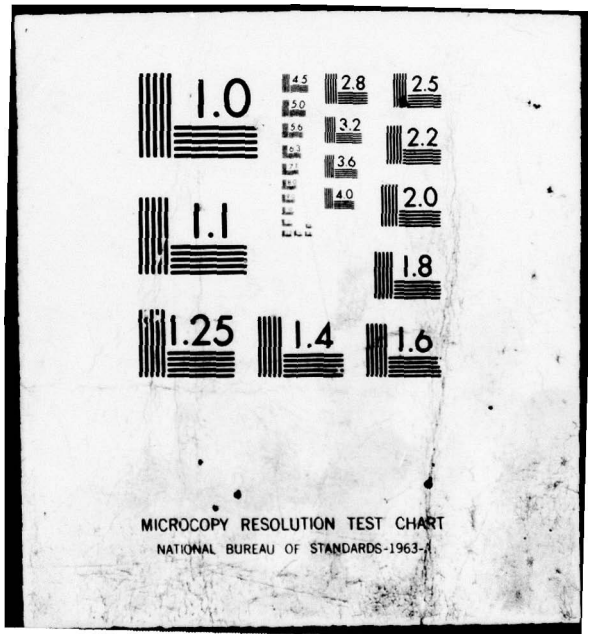
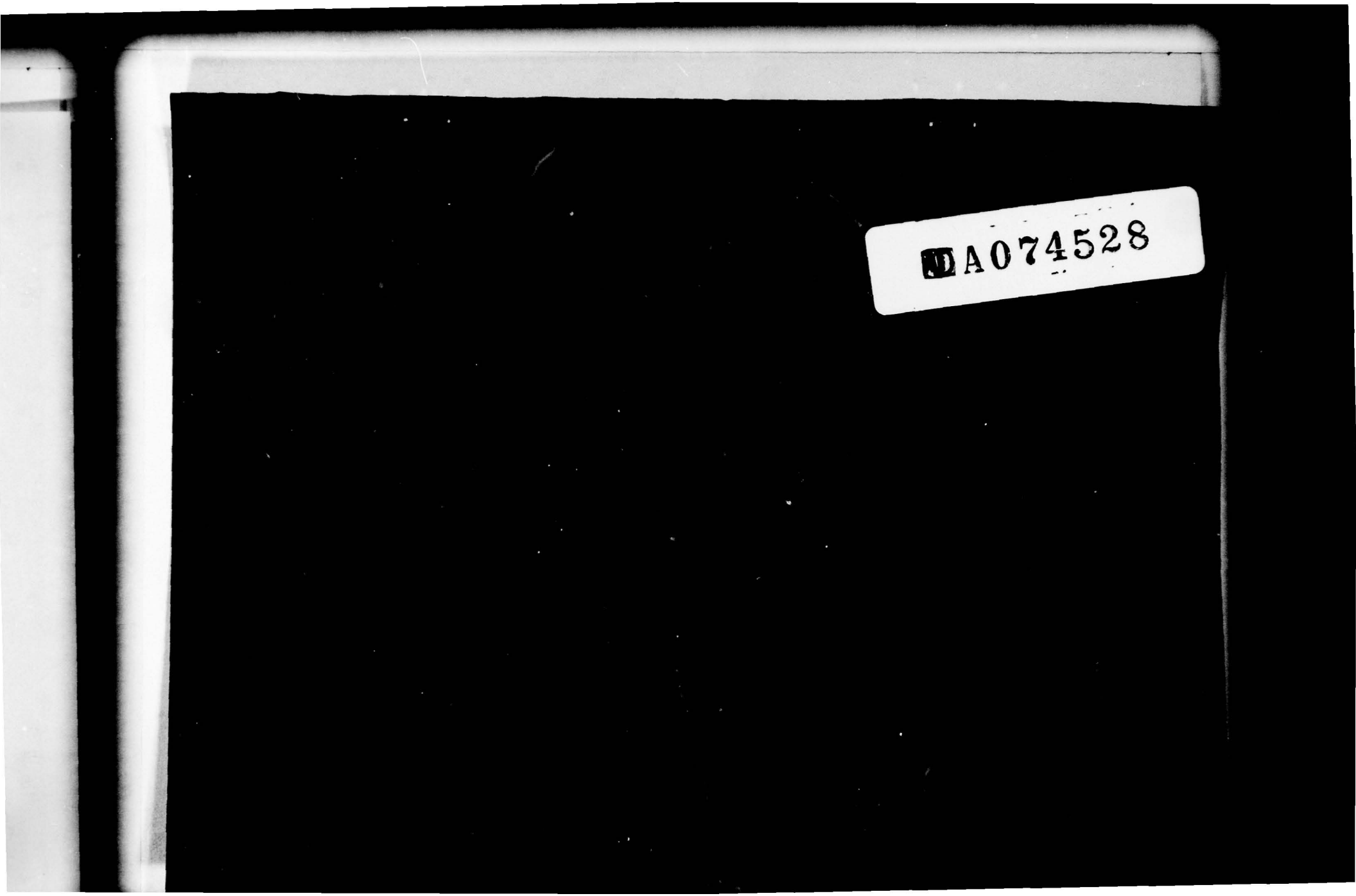
AD-A074 528 SCHOOL OF AEROSPACE MEDICINE BROOKS AFB TX F/G 9/2
WIZARD PROGRAMMER MANUAL.(U)
SEP 79 S D WILSON, M E WOMBLE
UNCLASSIFIED SAM-TR-79-15 NL

1 OF 1
AD A074528

END
DATE FILMED
11-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A



SAMPLE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SAM-TR-79-15	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* WIZARD PROGRAMMER MANUAL		5. TYPE OF REPORT & PERIOD COVERED Final Report Oct 1976 - Sep 1978
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Shelburne D. Wilson, Major, USAF, MC M. Edward Womble, Ph.D.		8. CONTRACT OR GRANT NUMBER*(s)* N/A
9. PERFORMING ORGANIZATION NAME AND ADDRESS USAF School of Aerospace Medicine (NGC) Aerospace Medical Division (AFSC) Brooks Air Force Base, Texas 78235		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62202F 7755-20-10
11. CONTROLLING OFFICE NAME AND ADDRESS USAF School of Aerospace Medicine (NGC) Aerospace Medical Division (AFSC) Brooks Air Force Base, Texas 78235		12. REPORT DATE Sep 1979 13. NUMBER OF PAGES 37
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Operating systems
Microcomputers

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

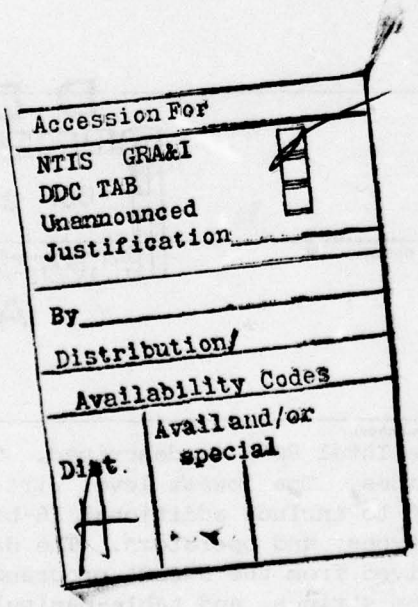
A string-oriented operating system for the Intel 8Ø8Ø is described. The system consists of a hierarchy of virtual machines. The lowest level virtual machines extend the instruction set of the 8Ø8Ø to include additional 16-bit arithmetic and logical instructions, new data types, and operators. The data types include strings and string operators derived from the SNOBOL programming language. A table data type is constructed from strings, and table-manipulation operators are provided. A bit-map data type and associated operators are also included. An Input/Output Control System (IOCS) supports device-independent

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

IO to multiple devices and diskette files. File-name aliases permit many logical IO streams to be dynamically mapped onto a restricted set of physical IO units. Pseudo device handlers expand the capabilities of IO devices and are transparent to application programs. Distributed command decoders interpret IO command strings. Once communication is established with a logical device, a low-overhead IO Vector mechanism may be used for further access. A keyboard monitor provides interactive debugging facilities to application programmers. System resource allocation is implementation dependent and is not embedded in the system nucleus. Multiple implementations over a range of system sizes have demonstrated the utility and adaptability of WIZARD.



CONTENTS

STRINGS

The WIZARD operating system is heavily string oriented. A WIZARD string consists of zero or more characters followed by an end-of-record character (EOR). WIZARD uses 7-bit ASCII characters with the high-order bit set to Ø. However, user programs may augment the standard character set by utilizing characters with the high-order bit set to 1. End-of-record characters are the Carriage Return (CR), Escape (ESC), Unit Separator (US), and Record Separator (RS). Except during input/output operations, WIZARD software considers these four EOR characters to be equivalent. User programs may select different record-termination characters to define various record types. In input/output operations, typical WIZARD device handlers treat CR and ESC as being equivalent record-termination characters of one type, and US and RS as equivalent record terminators of a second type. Output (or input echo) of CR or ESC causes execution of a CR and Ø or more line feeds; the number of line feeds is a user-settable handler parameter with a default value of 1. Output (or input echo) of US or RS terminates the input/output operation without causing carriage/cursor movement.

The use of multiple equivalent record-termination characters greatly increases user-program flexibility and allows for several different user-defined record types. Strings may be concatenated to construct higher order structures such as groups or files. Certain WIZARD routines utilize these higher order structures. By definition, a group is terminated by an EOR character immediately preceded by a Group Separator character (GS). Normally, this group terminator is implemented as a distinct two-character record. However, it is unsafe for a user program to search for the sequence "end-of-record, GS, end-of-record" or for GS as the initial character of a record. It is safe to search for GS as the character immediately preceding an EOR character. An end-of-file indicator is defined as a File Separator character (FS) immediately preceding an EOR character. It is unsafe to search for the sequence "EOR, FS, EOR" or for a file separator as the initial character of a record.

Although system routines in WIZARD version Ø normally manipulate strings in a unidirectional fashion from left to right, strings can be freely manipulated in a bidirectional fashion if an EOR character immediately precedes the first character in the record. Except for the initial record, this is automatically the case with concatenated strings.

ERRORS

The WIZARD error facility is used to service both system and user errors. All error messages and associated output are directed to the

logical device EO (Error Output). The default assignment for EO is the logical device CO (Console), but EO can be assigned to any other output device. If, however, EO is assigned to a logical device incapable of generating error messages, the system may crash: recursive error calls exhaust the available stack space and overwrite all memory. For example, if EO is assigned to a nonexistent diskette file, any error encountered will force a system crash.

To utilize the error facility from a user program, the address of the user-error message string is placed into register HL, the applicable error code is placed into register A, and a call to ERROR is executed. In WIZARD version Ø, error output consists of an error prefix followed by the user-error string. In addition, the WIZARD error code is displayed as well as the return address placed on the stack by the call to ERROR. It is thus possible to determine the location within user program from which the ERROR call was made. On return from ERROR, the failure flag (CARRY bit) is set. All other registers and flags are indeterminate.

FAILURE FLAG

The CARRY bit is used as a failure flag. Both system routines and application programs (user routines) set a failure flag to indicate that they were unsuccessful in carrying out the function requested. For example, a request to open a nonexistent diskette file would set CARRY at 1. In general, any error will set the failure flag. Failure is not necessarily undesirable. In many cases, routines are called in such a fashion that they may be expected to return a failure flag. The failure flag is true when CARRY is set at 1; the flag is false when CARRY is set at Ø.

FILE REFERENCE

A file reference identifies a particular file or group of files. File references may be either unique or ambiguous: A unique file reference identifies only one file, while an ambiguous file reference can be satisfied by several different files.

File references consist of four components: name--up to eight characters (NNNNNNNN); version--a period followed by up to three characters (.VVV); type--a colon followed by a single character (:T); and drive number--a slash followed by a digit between Ø and 3 (/2). The version, type, and drive components are optional and are set off from the name by their unique leading punctuation characters (NNNNNNNN.VVV:T/2). A missing name, version, or type is assumed to be blank, and a missing drive number is assumed to refer to the current default drive.

The following are examples of valid unambiguous file references:

MONITOR	MASTER/2	STARTREK.BAS/1
MONITOR.SRC	MASTER:$	STARTREK.XQT
MONITOR.OBJ:A	MASTER.ONE	STARTREK:Z/Ø

The special characters "?" and "*" may be used to make a file reference ambiguous so that it may match a number of different files. The "?" is used as a "wild-card" character which matches any character in the corresponding position in a file reference. Thus the ambiguous file reference PER????.BA? matches all of the following unambiguous file references:

PERFECT.BAL	PERSCI.BAS	PERQ.BAX

The character "*" is used to denote that all character positions to the right are wild-cards unless otherwise specified. The following examples illustrate the flexibility which this facility provides:

Reference	Equivalent to	Ambiguous reference matches
MONITOR.*	MONITOR.???:?	all files with name MONITOR
*.BAS	????????.BAS:?	all files with version .BAS
Z*	????????.???:?	all files starting with Z
*	????????.???:?	all files on the diskette

INPUT/OUTPUT CONTROL SYSTEM

Interface between user programs and files and input/output devices is through the WIZARD Input/Output Control System (IOCS). WIZARD provides two methods of access to input/output devices/files. The more general of these is the IOCS call, while the IO Vector call offers greatly reduced system overhead in accessing previously opened IO devices/files. IO devices and diskette files are logically equivalent, and unless specifically noted as an exceptional case, references to devices or files may be construed as being applicable to both. Systemwide register usage and linkage conventions are used to provide consistency in system access.

Basic Input/Output Functions

All IO device handlers are required to accommodate 11 basic IO functions: OPEN, CLOSE, IOCOM (IO command), CHRIN (character in), CHROUT (character out), STRIN (string in), STROUT (string out), BININ (binary in), BINOUT (binary out), BLKIN (block in), and BLKOUT (block out). In many cases, some of these functions may be inapplicable for a given device; if so, its handler must respond to the appropriate request in such a manner as to maintain system integrity. For example, a request for input from a line printer would normally result in an error message and the setting of a failure flag.

OPEN--All devices must be opened prior to use. (The exception is the console device, CO, which is opened by the system as part of its initialization procedure.) A device is normally assumed opened for input/output operations. When an unopened device is opened, all device parameters (line length, tab setting, etc.) are set to the default values. An OPEN issued to a device already opened normally causes no effect; however, some handlers may reset device parameters to their default values. An important function of the command OPEN is to return to the user program the IO handler-access code for use in subsequent low-overhead IO Vector access to the handler.

CLOSE--On completion of use, all IO devices must be closed. Although this is particularly important in the case of diskette files, for which failure to close will compromise both data integrity and proper system operation, it is also important for physical IO devices, many of which require end-of-file operations such as printer buffer clearing or tape trailer punching. An attempt to close an unopened device normally results in an error message. A multiply opened device may be closed as many times as it was previously opened; closing the console device (CO) is unnecessary.

IOCOM--This generalized input/output command is used by handlers for device-specific tasks such as setting device parameters (e.g., line length and tab stops) and for device-dependent operations (e.g., page eject and rewind). IO command requests are passed to the device handlers as IOCM strings.

CHRIN--Character In is used to obtain a single 7-bit ASCII character (high-order bit equals zero) in the accumulator.

CHROUT--Character Out is used to output a single 7-bit ASCII character from the accumulator.

STRIN--String In is used to input a single string of ASCII characters, placing them into a user-defined buffer.

STROUT--String Out is used to output a single string of ASCII characters from a user-defined buffer.

BININ--Binary In is used to input a single 8-bit byte of binary information into the accumulator.

BINOUT--Binary Out is used to output a single 8-bit byte of binary information from the accumulator.

BLKIN--Block In is used to input a block of 8-bit bytes of binary information of arbitrary user-defined length into a user-defined buffer.

BLKOUT--Block Out is used to output a block of 8-bit bytes of binary information of arbitrary user-defined length from a user-defined buffer.

Accessing Basic IO Functions Through IOCS

All basic input/output functions may be accessed through calls to IOCS. This access method is normally required for the command OPEN and is optional for the other basic IO commands. In all cases, calls to IOCS are made with the DE register pair pointing to the IOCS command string. Specific commands may require that the HL, A, and/or BC registers be loaded prior to the IOCS call. IOCS command strings consist of the name of the command to be executed, followed by the logical file name of the device to be accessed, followed by any command-specific parameters required. All fields in the IOCS command string are separated by one or more break characters (space, comma, tab) and are terminated by an EOR character, normally the CR.

OPEN--On this call to IOCS, the DE register points to the IOCS command string OPEN <file name>. The content of other registers is immaterial. On return from a successful OPEN, CARRY is set to Ø and DE contains the handler-access code to be used in IO Vector access to the open handler. The content of other registers is indeterminate. On return from an unsuccessful OPEN, CARRY is set to 1 and the content of all registers is indeterminate.

```
        LXI     D,OPNSTR
        CALL    IOCS
        . . .
OPNSTR: DB 'OPEN OUTPUT',CR
```

CLOSE--Prior to the call to IOCS, DE must point to the IOCS command string CLOSE <file name>. On return from a successful CLOSE, CARRY is set to Ø; if an error occurred on the attempt to close, CARRY is set to 1. Content of all other flags and registers is indeterminate.

```
        LXI     D,CLOSTR
        CALL    IOCS
        . . .
CLOSTR: DB 'CLOSE OUTPUT',CR
```

IOCOM--DE must be set pointing to the IOCS command string IOCM-- consisting of IOCOM, file name, and command specific argument string. In the example given, tab stops are to be set on the listing device at positions 5, 25, 50, and 75. On return, DE contains the handler-access code while the content of other registers is handler and command dependent.

```
        LXI     D,COMSTR
        CALL    IOCS
        . . .
COMSTR: DB 'IOCOM LO SETTAB 5,25,50,75',CR
```

CHRIN--DE must point to the IOCS command string CHRIN <file name>. On return, DE contains the handler-access code. The input character is found in A. If the operation was successful, CARRY is set to Ø; if unsuccessful, set to 1.

```
        LXI     D,CINSTR
        CALL    IOCS
        STA     NEWCHR
        . . .
CINSTR: DB 'CHRIN CO',CR
```

CHROUT--DE must point to the command string CHROUT <file name>, and A must contain the character to be output. On return, DE contains the handler-access code and A is normally unchanged; however, this is handler dependent. If the operation was successful, CARRY is set to Ø; if unsuccessful, set to 1. The content of other registers is not disturbed.

```
        LDA     OUTCHR
        LXI     D,COTSTR
        CALL    IOCS
        . . .
COTSTR: DB 'CHROUT CO',CR
```

STRIN--HL must point to the beginning of an input buffer to contain the input string. DE points to the command string STRIN <file name>. On return, DE contains the handler-access code, while HL points to the first location beyond the EOR character of the input string. BC is unchanged, A is indeterminate, and the flags are indeterminate except for CARRY which is set to Ø if the operation was successful, to 1 if unsuccessful.

```
        LXI     H,INBUF
        LXI     D,SINSTR
        CALL    IOCS
        . . .
SINSTR: DB 'STRIN CO',CR
```

STROUT--DE must point to the command string STROUT <file name>, and HL must point to the beginning of the string to be output. The output string must be terminated by an end-of-record character (CR, ESC, RS, US). On return, DE contains the handler-access code, while HL points to the first character beyond the end-of-record character terminating the output string. BC is unchanged; A and the flags are indeterminate except that CARRY is set to Ø if the operation was successful, to 1 if unsuccessful.

```
        LXI     H,OUTBUF
        LXI     D,SOTSTR
        CALL    IOCS
        . . .
SOTSTR: DB 'STROUT LO',CR
```

BININ--DE points to the command string BININ <file name>. On return, DE contains the handler-access code while the 8-bit binary input value is in the accumulator. HL and BC are unchanged; all flags are indeterminate except CARRY which is set equal to Ø if the operation was successful, to 1 if unsuccessful.

```
        LXI     D,BINSTR
        CALL    IOCS
        . . .
BINSTR: DB 'BININ LODFIL',CR
```

BINOUT--DE must point to the command string BINOUT <file name>, while the 8-bit binary value to be output is in the accumulator. On return, DE contains the handler-access code. The accumulator is normally unchanged, although this is handler dependent. HL and DE are unchanged. Flags are indeterminate except for CARRY which is set equal to Ø if the operation was successful, to 1 if unsuccessful.

```
        LXI     D,BOTSTR
        CALL    IOCS
        . . .
BOTSTR: DB 'BINOUT SAVFIL',CR
```

BLKIN--HL points to the beginning of the buffer into which the binary information will be input. BC contains a 16-bit binary number representing the maximum number of bytes to be input, and DE points to the command string BLKIN <file name>. On return, DE contains the handler-access code. HL points to the memory location following the last byte input; BC is Ø if all requested bytes were input. If fewer bytes were input than requested, BC contains a 16-bit binary number representing the number of requested bytes that were not input. Content of the accumulator and flags is indeterminate except that CARRY is set to Ø if the operation was successful, to 1 if unsuccessful.

```
        LXI     H,INBUF
        LXI     B,COUNT
        LXI     D,BKISTR
        CALL    IOCS
        . . .
BKISTR: DB 'BLKIN LODFIL',CR
```

BLKOUT--HL points to the first byte of memory to be output. BC contains a 16-bit binary number giving the maximum number of bytes to be output. DE points to the command string BLKOUT <file name>. On return, HL points to the memory location following the last byte output, and BC contains a 16-bit binary number giving the number of output bytes which could not be successfully output. This value is normally Ø. DE contains the handler-access code.

```
LXI       H,OUTBUF
LXI       B,COUNT
LXI       D,BKOSTR
CALL      IOCS
. . .
BKOSTR:   DB 'BLKOUT SAVFIL',CR
```

Input/Output Vector Calls

Although the WIZARD IOCS call provides great flexibility in accessing IO devices and diskette files with logical file names, it does so at the expense of significant overhead. The overhead required to access an open device/file can be greatly reduced by using IO Vector calls to the basic IO functions. Each of the 11 basic IO functions has an associated IO Vector call. Prior to using any IO Vector call, the device/file's handler address (device-access code) must be placed into DE (returned in DE by the IOCS OPEN routine). All basic IO Vector calls return with the device/file's handler address unchanged in DE, which allows successive IO Vector calls to be made without reloading DE. Certain IO commands require that other registers be set prior to call.

OPEN--In systems on which the complete IOCS has been implemented, the IO Vector routine OPEN must not be used because device/file's handler addresses are subject to change between successive OPENs of the same device. However, in some minimal hardware configuration systems in which the full IOCS is not implemented, the IO Vector routine may be used. In this case, the programmer must know the (fixed) handler address. On the unusual occasions where the IO Vector call to OPEN is appropriate, the device-access code is known at the time the program is written; therefore, an LXI instruction is appropriate to load the DE register pair. Content of registers other than DE is immaterial. On return from a successful OPEN, the CARRY flag will be set to Ø and DE will contain the device/file's handler address to be used in IO Vector access to the open handler. The content of other registers is indeterminate. On return from an unsuccessful OPEN, CARRY is set to 1 and the content of all registers is indeterminate.

```
LXI       D,DEVCOD
CALL      OPEN
```

CLOSE--At the call to CLOSE, DE contains the device-access code. On return from CLOSE, content of all registers and flags is indeterminate, except that if an error occurred on the attempt to close, CARRY is set to 1; otherwise, set to Ø.

```
XCHG              ;SAVE HL
LHLD  DEVCOD      ;PICK UP STORED DEVICE CODE
XCHG              ;DEVICE CODE TO DE, RESTORE HL
CALL CLOSE
```

IOCOM--For this call, DE must contain the device-access code and HL must point to the command-specific argument string. (Note that this argument string is identical to the argument string portion of the IOCOM command string used when accessing IOCOM through IOCS.) On return, DE contains the handler-access code, while the content of other registers is handler and command dependent. In the example given, tab stops are to be set on the listing device at positions 5, 25, 50, and 75.

```
        LHLD    DEVCOD
        XCHG
        LXI     H,COMSTR
        CALL    IOCOM
        . . .
COMSTR: DB 'SETTAB 5,25,50,75',CR
```

CHRIN--DE must contain the handler-access code. On return, DE still contains the handler-access code, and the input character is found in A. If the operation was successful, CARRY is set to Ø; if unsuccessful, set to 1.

```
        LHLD    DEVCOD
        XCHG
        LXI     H,INBUF
        CALL    CHRIN
```

CHROUT--DE must be loaded with the handler-access code and A must contain the character to be output. On return, DE is unchanged; A is normally unchanged but is handler dependent. If the operation is successful, CARRY is set to Ø; if unsuccessful, set to 1. The content of other registers is not disturbed.

```
        XCHG            ;SAVE PNTR TO DATA
        LHLD DEVCOD     ;PICK UP DEVICE CODE
        XCHG            ;DEVICE CODE TO DE,RESTORE PNTR
        MOV A,M         ;PICK UP OUTPUT CHAR
        CALL CHROUT     ;OUTPUT THE CHAR
```

STRIN--DE must be loaded with the handler-access code. HL points to the beginning of an input buffer to contain the input string. On return, DE still contains the handler-access code, while HL points to the first location beyond the end-of-record character of the input string. BC is unchanged; A is indeterminate. Flags are indeterminate except that CARRY is set to Ø if the operation was successful, to 1 if unsuccessful.

```
        LHLD    DEVCOD
        XCHG
        LXI     H,INBUF
        CALL    STRIN
```

STROUT--DE contains the handler-access code, while HL points to the beginning of the string to be output. The output string must be terminated by an end-of-record character. On return, DE still contains the handler-access code, while HL points to the first character beyond the end-of-record character terminating the output string. BC is not disturbed. A and the flags are indeterminate except that CARRY is set to Ø if the operation was successful, to 1 if unsuccessful.

```
LHLD    DEVCOD
XCHG
LXI     H,OUTBUF
CALL    STROUT
```

BININ--DE must contain the handler-access code. On return, DE still contains the handler-access code, while the 8-bit binary input value is in the accumulator. HL and BC are unchanged. All flags are indeterminate except CARRY, which is set to Ø if the operation was successful, to 1 if unsuccessful.

```
LHLD    DEVCOD
XCHG
CALL    BININ
```

BINOUT--DE contains the handler-access code, while the 8-bit binary value to be output is in the accumulator. On return, DE still contains the handler-access code; the accumulator is normally unchanged although this is handler dependent. HL and DE are unchanged. Flags are indeterminate except that CARRY is set to Ø if the operation was successful, to 1 if unsuccessful.

```
XCHG
LHLD    DEVCOD
XCHG
MOV     A,M
CALL    BINOUT
```

BLKIN--DE contains the handler-access code. HL points to the beginning of the input buffer into which the binary input information will be placed. BC contains the 16-bit binary number representing the maximum number of bytes to be input. On return, DE still contains the handler-access code; HL points to the memory location following the last byte input; and BC is zero if all requested bytes were input. If the number of bytes input was less than the number requested, BC contains a 16-bit binary number representing the number of requested bytes that were not input. Content of the accumulator and flags is indeterminate except that CARRY is set to Ø if the operation was successful, to 1 if unsuccessful.

```
LHLD    DEVCOD
XCHG
LXI     B,COUNT
LXI     H,INBUF
CALL    BLKIN
```

BLKOUT--DE contains the handler-access code, HL points to the first byte of memory to be output, and BC contains a 16-bit binary number giving the maximum number of bytes to be output. On return, DE still contains the handler-access code, HL points to the memory location following the last byte output, and BC contains a 16-bit binary number giving the number of output bytes that could not be successfully output. This value is normally Ø.

IOCS DISK AND FILE MANAGEMENT

File Access Table

WIZARD maintains an internal file access table (FAT) to associate logical file names with physical devices and disk files. When the system is powered up or reset, a set of default file names is entered into the file access table. This default table can be modified by the DECLARE and ASSIGN functions. A logical file name must be entered into the file access table before it can be referenced by the system.

Disk/File Management Commands

DECLARE--Used to associate a user-defined name with a disk file or physical device handler. This function is also used to delete files from the file access table.

```
DECLARE   <FILNAM>  NULL
                    FILE
                    DEVICE @<ADDR>
LXI       D,DCLSTR
CALL      IOCS
. . .
DCLSTR:   DB 'DECLARE INFILE FILE',CR
```

A file name is declared to the system as a disk file with the command string DECLARE <FILNAM> FILE. A file name may be associated with a physical device handler with the statement DECLARE <FILNAM> DEVICE@<ADDR>. The address is the hexadecimal value of the base address of the device handler. All device handlers specified in this manner must conform to the WIZARD device-handler specifications. Any file name that has been entered into the file access table by the DECLARE or ASSIGN commands may be deleted from the table by being declared NULL. An attempt to nullify a nonexistent file results in an error message, but does not compromise system integrity. Already existing file names may be redeclared at any time, in which case the previous declaration is superseded. In all file names, the diskette drive designator (e.g. /Ø) is part of the file name.

ASSIGN--Used to associate a new file name with a previously defined file name.

```
        LXI     D,ASNSTR
        CALL    IOCS
        . . .
        ASNSTR:   DB 'ASSIGN OUTPUT TO LO',CR
```

In the example given, the logical file OUTPUT is associated with the predefined name LO (Listing Out). Subsequent references to the logical file OUTPUT are directed to LO. ASSIGN may be used to modify input/output assignments at run time.

CREATE--Interfaces to the PerSci intelligent diskette controller and is used to place file names into the diskette directory and to reserve storage for the file.

```
        CREATE    <FILNAM> <FILSIZE>
        LXI       D,CRESTR
        CALL      IOCS
        . . .
        CRESTR:  DB'CREATE TMPFILE 40',CR
```

The file name in the CREATE command string must be a legal disk file name (see File Reference section) and must not already exist on the diskette directory. The file-size field in the command string determines how many 128-byte diskette sectors are to be reserved for the file. It must be an integer value greater than Ø. The command CREATE does not utilize the file access table, and the name of the file to be created need not be declared prior to creation.

DELETE--Used to delete diskette files.

```
        DELETE    <FILNAM>
        . . .
        LXI       D,DELSTR
        CALL      IOCS
        . . .
        DELSTR:   DB'DELETE TMPFILE',CR
```

If an attempt is made to delete a nonexistent file, an error condition results but system integrity is not compromised. The DELETE command does not utilize the file access table, and the file name need not be declared prior to deletion. Deletion of a file does not remove that file name's file access table entry, if any.

COPY--Used to copy one disk file to another.

```
        COPY      <OLDFIL> TO <NEWFIL>
        . . .
        LXI       D,CPYSTR
        CALL      IOCS
        CPYSTR:   DB'COPY */Ø to */1',CR
```

Under current implementation, COPY can be used only to transfer information from one diskette file to another. A new file is created and information in an old file is copied to it. The original (old) file remains unaltered. The COPY command does not utilize the file access table; neither the old file nor the new file need be declared. The new file must not already exist on the destination diskette or a system error will result; the file already existing will not be altered. Due to restrictions imposed by the PerSci intelligent disk controller, all diskette files must be closed before the COPY command is issued. In the example given, contents of the system diskette are transferred into the user diskette.

RENAME--Used to change the name of a diskette file or group of diskette files. In the example, disk file named MASTER is renamed OLDMASTR.

```
        RENAME    <OLDNAM> TO <NEWNAM>
        . . .
        LXI       D,NAMSTR
        CALL      IOCS
        . . .
        NAMSTR:   DB 'RENAME MASTER TO OLDMASTR',CR
```

The name of a file in the diskette directory is changed without modifying the contents of the file. The renamed command does not utilize the file access table, and neither the old name nor the new name need be declared. The new file name must not be the same as any name already on the disk.

DIRECTORY--Used to obtain diskette directory information.

```
        DIRECTORY <FILREF>
        LXI       B,DIRBUF
        LXI       D,DIRSTR
        CALL      IOCS
        DIRSTR:   DB'DIRECTORY */Ø',CR
```

The command must be spelled out and may not be abbreviated. The default-file reference is the user diskette (*/1). The system diskette directory may be obtained using a file reference of */Ø. If directory information on only a specific file is desired, that file name should appear as a file reference. Prior to the call to DIRECTORY, DE must point to the DIRECTORY command string while BC must point to a user buffer to contain the directory. The DIRECTORY command places one or more strings into the user-defined buffer, terminating with a group-separator string (GS,CR).

FORMAT--Used to initialize a diskette to the IBM soft-sectored format.

```
        FORMAT    DISKNAM
        . . .
        LXI       D,FMTSTR
        CALL      IOCS
        . . .
        FMTSTR:   'FORMAT SCRATCH',CR
```

The argument of the command FORMAT is written onto the new diskette as the diskette name.

INTERFACE TO FLOPPY DISK SUBSYSTEM

WIZARD version Ø, modification level Ø through 2, utilizes the PerSci intelligent diskette controller. Many characteristics of the diskette interface are dictated by the requirements and characteristics of the disk controller. Although the capabilities of the PerSci controller are generally adequate and its performance predictable, some conditions (especially error conditions) encountered in system use result in effective loss of communication between the WIZARD operating system and the intelligent controller. If disk accesses result in inappropriate disk-error messages, it may be desirable to close all disk files known to be open and to issue a CLRDSK command to clear any spuriously open disk files. If proper diskette operation is still not obtained, it may be necessary to power down the system and restart.

Disk-Handler/File-Handler Relationships

All input/output operations involving diskette files are mediated by the WIZARD file handler. Up to five diskette files may be opened simultaneously. Diskette files are logically equivalent to distinct IO devices. The WIZARD file handler and its associated pseudohandlers accomplish the physical disk IO by invoking the disk handler. User programs should not attempt to do standard input/output operations directly to the disk. A variety of disk utility functions, however, may be invoked directly from user programs. These functions are explicitly identified in the next section. Due to characteristics of the PerSci intelligent diskette controller, the behavior of standard IO functions is slightly different when referencing disk files rather than physical IO devices. These differences are minimal, however, and are not normally apparent to user programs. It is important to recognize that the association of diskette files with handler base addresses, which occurs when a file is opened, does not persist after the file has been closed. If the file is subsequently reopened, it very probably will be associated with a different handler base address.

Basic File Input/Output Commands

OPEN--This command opens the diskette file for use. The diskette file name must have been declared to be a file prior to an attempt to open the file. All files are assumed to be opened for read, write, and update. OPEN will fail if the disk drive is not ready or if the requested file is not on the disk. An attempt to open an already opened file is ignored and is not considered to be an error condition.

CLOSE--An open file must be closed by this command. Failure to close a disk file may compromise the integrity of the file and may cause unreliable system operation. An attempt to close an already closed disk file is an error. The handler base address, which is associated

with a file by the command OPEN, ceases to be effective after the command CLOSE.

CHRIN--The Character-In command reads a 7-bit ASCII character (high-order bit equals Ø), returning the result in the accumulator. No attempt is made to check with parity of the character as actually written on the disk. If CHRIN is used to read a file generated by STROUT, any end-of-record character other than a record separator (RS) may be expected to be followed by an RS. This is due to a characteristic of the PerSci intelligent disk controller.

CHROUT--The Character-Out command is used to write an ASCII character passed in the accumulator to the disk. The character is written on the disk as an 8-bit byte with an odd parity. If strings are written to the disk using CHROUT and are subsequently read with the string-in command (STRIN), the results may be unpredictable due to idiosyncrasies of the PerSci intelligent disk controller.

STRIN--If STRIN is used to read records that were not generated by the string-out command (STROUT), the results may be unpredictable due to the idiosyncrasies of the PerSci intelligent disk controller. Characters input by STRIN are 7-bit ASCII, with the high-order bit set equal to Ø irrespective of the presence or absence of parity bits on the diskette.

STROUT--This command appends supernumerary record-separator characters at the end of strings terminated by an end-of-record character other than RS. This is due to STROUT's use of the punctuated access capability of the PerSci intelligent disk controller. These extra RS characters are stripped out on input by the STRIN routine and are required for its proper operation. STROUT writes characters to the disk as 8-bit binary values with odd parity.

BININ--Binary In reads a full 8-bit byte from the disk, returning the result in the accumulator. When used to read information generated by STROUT, extra RS characters may appear following other end-of-record characters.

BINOUT--Binary Out writes an 8-bit byte passed from the accumulator to the disk.

BLKIN--Block In reads a block of binary bytes. If used to read information written to the disk by STROUT, extra RS characters may be embedded in the data.

BLKOUT--Block Out is used to write a block of binary 8-bit bytes to the disk.

IOCOM--Disk and file IOCOMs are divided between those that reference specific diskette files and are accessed through reference to those files and those that are directed to the disk handler per se and are referenced to the logical device DSK.

DIRECTORY--This IO command is used to obtain directory information on one or more diskette files and is directed to DSK. Its argument is a file reference. If no file reference is given, the directory information returned is that for the default disk per se. If a file reference is given, the information consists of the directory information for the diskette per se plus directory information on all files satisfying the file reference. The appropriate file reference to obtain the entire directory for the system diskette is */Ø, and for the user diskette is */1. On a call to DIRECTORY, BC points to a user buffer where the directory will be placed. The directory may consist of multiple strings and is terminated by a group-separator string. An out-of-space error encountered while doing a directory indicates a damaged disk.

COPY--This command implements a disk to disk-file copy. It should normally be invoked only through IOCS. All files must be closed before the COPY command is issued. This requirement is imposed by the characteristics of the PerSci intelligent disk controller.

CREATE--This command, used to allocate space for a disk file and to create a directory entry, should normally be invoked only through IOCS.

DELETE--This command, used to delete a disk file from the diskette, should normally be invoked only through IOCS.

RENAME--This command, used to rename a diskette file, should normally be invoked only through IOCS.

FORMAT--Directed to DSK, this command is used to write format information to a new diskette. It may also be used to regenerate a diskette that has been damaged through a hardware failure which causes persistent hard-disk errors. The argument to FORMAT is the diskette name that will be written into the diskette directory. The FORMAT command forces generation of a diskette with optimum sector interleave to maximize diskette capacity. Complete execution of the FORMAT command requires several seconds.

GAP--By moving files so as to eliminate unused space, this command compacts data on a diskette. The names of all files moved are written to the console device. GAP is directed to the logical device DSK and should normally be used after diskette files are deleted. There is no argument, and the user disk is assumed. All disk files must be closed before the command GAP is issued. This requirement is imposed by the PerSci intelligent disk controller.

DSKDX--This command runs a diagnostic test routine on the diskette drive and is directed to the logical device DSK. Its argument is either I, R, or V, depending on the diagnostic routine desired. If the argument is I, an increment and test routine will be executed and control will be returned to the user. If R or V is used, random access tests are made and control is not returned to the user. Regaining system control requires restarting the system. The integrity of disk data may be compromised by using this diagnostic command.

CLRDSK--The Clear-Disk command is used to insure that all disk files are closed. It is normally used when an error condition results in a disagreement between the WIZARD file-management routine and the PerSci intelligent disk controller as to which files are open. This command is directed to the logical device DSK and is automatically issued on system restart.

POSITN--The Position command is used to set or report current file pointer location in bytes relative to the beginning of the file. It is directed to an open diskette file. If the argument is a question mark, the current location is reported. (BC is used as a pointer to a user-defined buffer area to contain the result.) If the argument is an unsigned decimal number, the file is positioned to that byte. If the argument is a signed (+,-) decimal number, the file pointer is moved relative to its previous location by number of bytes indicated in either a + or - direction. All arguments to and results returned by the Position command are in the form of decimal ASCII strings. If a request is made to position a file beyond the end of file, the pointer is moved to the end of file and there is no error indication.

BAKSPA--The Backspace command is used to move the file pointer backwards until the beginning of a record is encountered. It is directed to a previously opened diskette file. If the file pointer is pointing to the beginning of a record when BAKSPA is invoked, it will be moved backwards to point to the beginning of the preceding record. If pointing to the interior of a record when this command is invoked, the file pointer is moved back to the beginning of that record. This command involves very substantial overhead and should not be used injudiciously.

SKIPEOF--The Skip EOF command is used to move the file pointer to the physical end of file. It works correctly only with a file of 511 blocks or less.

REWIND--This command is used to position the file pointer at the beginning of an open file.

DATE--This command is used to inform the PerSci intelligent disk controller of the current date and default drive. It is directed to the logical device DSK, and its argument is a string containing the date in the format YYMMDD/default drive number. DATE is not normally invoked by user programs.

Disk and File Errors

An attempt to perform IO operations on an unopened file results in a File-Not-Opened error. This condition is detected by the file-handler routines; no communication is established with the PerSci intelligent disk controller. An attempt to invoke a nonexistent disk/file IOCOM results in a nonexistent disk IOCOM error; system and data integrity are maintained.

Two types of error may be returned from the PerSci intelligent disk controller. Soft disk errors (recoverable read/write errors) are identified to the error device, E0, but do not cause the failure flag to be set and do not otherwise influence system operation. If after five retries a soft disk error persists, a hard-error message is produced and the failure flag is set. All other disk errors are comparable to hard disk errors in that they result in setting of the failure flag and abortion of the current command.

The error message test is generated by the PerSci intelligent disk controller. Due to timing considerations at the interface to the PerSci intelligent disk controller, occasionally the initial character of a disk-error message may be dropped. If a disk error is encountered, appropriate recovery procedure for the user program is to attempt to close the disk file on which the error occurred. If this results in a second error message, no harm has been done. However, leaving a disk file open under these circumstances is unacceptable as the PerSci intelligent disk controller may be caused to exhibit unpredictable behavior.

WIZARD UTILITIES

The WIZARD operating system is constructed around a virtual machine that, using a collection of utility subroutines, is implemented on the 8080 processor. These subroutines may occupy approximately 2 kilobytes of read-only memory and are fully accessible to user programs. The utility subroutines are divided into five general classes, comprised of basic string manipulation, numerical conversion, table manipulation, extended arithmetic and logic, and bit-map manipulation.

String Manipulation

The WIZARD utility-subroutines package contains a number of routines that implement string manipulation primitives. These functions are largely based on primitives of the language SNOBOL. The HL-register pair contains a pointer to the subject string being manipulated. A number of string manipulation routines use only this register pair. Pattern-matching routines use the DE-register pair to point to the beginning of

the pattern. Specific-register usage is contained in the description of the individual routines. As a general rule, however, on call, HL points to the location and the subject string at which pattern matching or other string manipulation is to begin. This pointer may be moved according to the specifications of the various routines but will under no circumstances be moved beyond the end-of-record character. On return, HL points to the first character not matched in pattern-matching operations. On call to pattern-matching routines, DE points to the beginning of the pattern string; and on return, is indeterminate. Except as specifically noted in the individual routine descriptions, the BC-register pair is not used and is returned unchanged. Except as specifically noted, the values of the A register and the flags are immaterial at the time of call and indeterminate on return, with the exception of the flag CARRY which is set equal to 1 to indicate failure. Failure conditions for the individual routines are indicated in their descriptions.

PATTERNS--Basic string-matching patterns: Unlike all of the other WIZARD utilities, this is not a subroutine. It is, rather, a collection of commonly used pattern strings that is made available to system and user programs in order to avoid unnecessary duplication of commonly used patterns.

BRKCHR--The system's standard break characters used to separate fields within strings are the space, comma, and tab characters.

NUMBRS--Decimal digits Ø through 9.

CAPS--Capital letters of the ASCII character set A through Z.

LOCASE--The lowercase ASCII letters a through z.

LETRS--The uppercase ASCII letters A through Z followed by lowercase ASCII letters a through z. Note: WIZARD uses the standard ASCII collating sequence.

ALPH--The numbers Ø through 9 followed by uppercase letters A through Z, followed by lowercase letters a through z.

PUNT--The printing ASCII characters, excluding numeric digits and letters in ascending collating sequence.

ANY--Matches at most one character in the subject string with one in the pattern string. It succeeds if any character in the pattern string matches the character in the subject string pointed to by HL; it fails if either the subject string or the pattern string is null. If ANY succeeds, HL is incremented to point to the first character following the subject-string character matched; if it fails, HL is unchanged. DE is not preserved.

NOTANY--Matches at most one character in the subject string. It succeeds if the subject-string character pointed to by HL is not matched by any character in the pattern string; it fails if either the subject string or the pattern string is null. If NOTANY succeeds, HL is incremented to point to the first character following the subject-string character matched. If NOTANY fails, HL is unchanged. DE is not preserved.

SPAN--Matches the longest contiguous string of characters in the subject string, starting with the location pointed to by HL, with one also found in the pattern string. A null subject or pattern string fails. SPAN leaves HL pointing to the first character not matched.

BREAK--Searches for the longest contiguous string of characters in the subject string, starting at the location pointed at by HL, that is not matched in the pattern string. A null pattern or subject string fails. BREAK returns with HL pointing to the first character not matched. DE is not preserved.

MATCH--Matches the subject string with pattern string, character for character, to the end of the pattern. MATCH fails if the two strings do not match, character for character, to the end of the pattern. Pattern is terminated by an EOR character. A null pattern succeeds. If successful, MATCH leaves HL pointing to the first unmatched character; if unsuccessful, HL is unchanged. DE is not preserved.

BRKMAT--Matches the subject string with pattern string, character for character, to the end of the pattern. BRKMAT fails if the two strings do not match, character for character, to end of pattern. The pattern is terminated by an EOR character, break character (space, comma, tab), or exclamation point. A null pattern succeeds. If successful, BRKMAT leaves HL pointing to the first unmatched character; if unsuccessful, HL is unchanged. DE is not preserved.

MOVSTR--Moves the string pointed to by DE to the location pointed to by HL. A string is terminated by any EOR character. Upon return, DE points to the EOR character terminating the source string; HL points to the EOR terminating the destination string. If the string destination address at the time of call is between the source-string address and the EOR character terminating the source string, then MOVSTR will destroy the contents of all of the system read/write memory and will crash the system.

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	DESTINATION ADDRESS	POINTS TO SOURCE EOR
HL	SOURCE ADDRESS	POINTS TO DESTINATION EOR
A	IMMATERIAL	INDETERMINATE
FLAGS	IMMATERIAL	INDETERMINATE

MOVSTR does not fail.

BRKMOV--Moves the string to the first break character (space, tab, comma) or EOR character. The terminating character (either break or end-of-record) is not moved. BRKMOV leaves DE pointing to the first source-string character not moved and HL pointing to the first destination string character available (not moved). BRKMOV functions identically to MOVSTR except that it moves a string or substring terminated by a break character or an EOR character, and it leaves DE and HL pointing to the first character not moved rather than the last character moved. BRKMOV does not fail.

LEN--Returns the length of a string. On call, HL points to the string; the length of the string is returned in DE as a 16-bit binary number. The EOR character is not included in the length. LEN changes only DE. LEN does not fail.

REM--Moves pointer from the beginning of a string to its end. On call, HL points to the subject string. On return, HL points to the EOR character terminating the string. Other registers are unchanged. REM never fails.

NXTARG--Obtains the next argument in an argument string. Arguments are separated by one or more break characters (tab, space, comma). NXTARG is called with HL pointing to the current location in the subject string; it returns with HL pointing to the first character in the next argument. The program status word is destroyed. NXTARG fails if an EOR character is encountered.

Numerical Conversion Routines

This collection of utility routines is used for converting numerical information between the alternate forms of unsigned binary (4 or 16 bits, depending on the routine) and ASCII strings using any of three number-base systems (octal, decimal, hexadecimal).

HEXIT--Converts the low-order 4 bits (low-order nybble) in the accumulator to an ASCII hexadecimal character in the accumulator.

REGISTER	ON CALL	ON RETURN
A	ARGUMENT IN LOW-ORDER NYBBLE	ASCII HEXADECIMAL CHARACTER
BC	IMMATERIAL	UNCHANGED
DE	IMMATERIAL	UNCHANGED
HL	IMMATERIAL	UNCHANGED
FLAGS	IMMATERIAL	INDETERMINATE

HEXIT does not fail.

DEOCT--Accepts a legal octal digit as an ASCII character in the accumulator and returns a binary equivalent in the 4 low-order bits of the accumulator. If the argument is not a legal octal digit (Ø through 7), the accumulator is set to Ø and the routine fails.

REGISTER	ON CALL	ON RETURN
A	ASCII DIGIT	BINARY VALUE Ø THROUGH 7
BC	IMMATERIAL	UNCHANGED
DE	IMMATERIAL	UNCHANGED
HL	IMMATERIAL	UNCHANGED
FLAGS	IMMATERIAL	INDETERMINATE EXCEPT CARRY: CARRY SET AT Ø IF DEOCT SUCCEEDS; AT 1 IF IT FAILS

DEDEC--Accepts a legal decimal digit as an ASCII character in the accumulator and returns the binary equivalent in the accumulator. If the argument passed in the accumulator is not a legal decimal digit (Ø through 9), the accumulator is set to Ø and the routine fails.

REGISTER	ON CALL	ON RETURN
A	ASCII DIGIT	BINARY DIGIT
BC	IMMATERIAL	UNCHANGED
DE	IMMATERIAL	UNCHANGED
HL	IMMATERIAL	UNCHANGED
FLAGS	IMMATERIAL	INDETERMINATE EXCEPT CARRY: CARRY SET AT Ø IF DEDEC SUCCEEDS; AT 1 IF IT FAILS

DEHEX--Accepts a legal hexadecimal digit as an ASCII character in the accumulator and returns the binary equivalent in the accumulator. If the character passed in the accumulator is not a legal hexadecimal digit (Ø through F), the routine fails.

REGISTER	ON CALL	ON RETURN
A	ASCII DIGIT	BINARY DIGIT
BC	IMMATERIAL	UNCHANGED
DE	IMMATERIAL	UNCHANGED
HL	IMMATERIAL	UNCHANGED
FLAGS	IMMATERIAL	INDETERMINATE EXCEPT CARRY: CARRY SET AT Ø IF DEHEX SUCCEEDS; AT 1 IF IT FAILS

GETHEX--Converts a hexadecimal ASCII substring to a 16-bit binary number. GETHEX is called with HL pointing to the first ASCII hexadecimal character to be converted and returns with HL pointing to the first nonhexadecimal character. The unsigned 16-bit binary result is returned in DE. If the first ASCII character is not a legal hexadecimal character (Ø through F), DE is set to Ø and the routine fails. If the first ASCII character in the subject string is a legal hexadecimal character, DE is set equal to the binary equivalent (low-order 16 bits) of the argument string and fail is set to false (CARRY set at Ø).

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	IMMATERIAL	16-BIT BINARY RESULT
HL	POINTS TO FIRST HEXADECIMAL CHARACTER	POINTS TO FIRST NONHEXADECIMAL CHARACTER
A	IMMATERIAL	INDETERMINATE
FLAGS	IMMATERIAL	INDETERMINATE EXCEPT CARRY: CARRY SET AT Ø IF GETHEX CALLED WITH AT LEAST ONE LEGAL CHARACTER; SET AT 1 IF CALLED WITH NO LEGAL CHARACTER

GETOCT--Converts an octal ASCII string to a 16-bit binary number. GETOCT is called with HL pointing to the first ASCII octal character and returns with HL pointing to the first nonoctal character. The unsigned 16-bit binary result is returned in DE. If the first character is not a legal octal numeric (Ø through 7), DE is set to Ø and the routine fails. Otherwise, DE is set to the unsigned binary equivalent (low-order 16 bits) of the argument string, and fail is set to false.

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	IMMATERIAL	16-BIT BINARY RESULT
HL	POINTS TO FIRST OCTAL CHARACTER	POINTS TO FIRST NONOCTAL CHARACTER
A	IMMATERIAL	INDETERMINATE
FLAGS	IMMATERIAL	INDETERMINATE EXCEPT CARRY: CARRY SET AT Ø IF AT LEAST ONE LEGAL CHARACTER; SET AT 1 IF NO LEGAL CHARACTERS

GETDEC--Converts decimal ASCII string to 16-bit binary. GETDEC is called with HL pointing to the first ASCII decimal and returns with HL pointing to the first nondecimal character. The unsigned 16-bit binary result is returned in DE. If the first character of the argument string is not a legal decimal digit (Ø through 9), DE is set equal to Ø and the routine fails. If the first character of the argument string is a legal decimal digit, DE is set equal to the unsigned 16-bit binary equivalent (low-order 16 bits) of the argument string, and fail is set equal to false.

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	IMMATERIAL	16-BIT BINARY RESULT
HL	POINTS TO FIRST DECIMAL CHARACTER	POINTS TO FIRST NONDECIMAL CHARACTER
A	IMMATERIAL	INDETERMINATE
FLAGS	IMMATERIAL	INDETERMINATE EXCEPT CARRY: CARRY SET AT Ø IF AT LEAST ONE LEGAL CHARACTER; SET AT 1 IF NO LEGAL CHARACTER

TODEC--Converts a 16-bit unsigned binary integer in DE to a decimal ASCII string value in memory. The created string consists of exactly 5 decimal digits with leading zeros.

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	16-BIT INTEGER	INDETERMINATE
HL	POINTS TO FIRST DECIMAL DIGIT	POINTS TO LOCATION IMMEDIATELY FOLLOWING FIFTH DECIMAL DIGIT
PSW	IMMATERIAL	INDETERMINATE

TODEC does not fail.

SPADEC--Converts a 16-bit unsigned binary integer in DE to an ASCII decimal string in memory. The string produced is exactly 5 characters in length, right-justified with leading zeros replaced by blanks.

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	16-BIT INTEGER	INDETERMINATE
HL	POINTS TO FIRST DIGIT OF RESULT	POINTS PAST LAST DECIMAL DIGIT
PSW	IMMATERIAL	INDETERMINATE

SPADEC does not fail.

COMDEC--Converts a 16-bit unsigned binary integer in DE to an ASCII decimal string in memory. The result consists of 1 to 5 digits. COMDEC may disturb characters in the result buffer following the decimal digits it generates. In all cases, it returns with HL pointing to the first memory location following the low-order decimal digit produced. In no case will more than five memory locations be generated and/or disturbed.

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	16-BIT INTEGER	INDETERMINATE
HL	POINTS TO FIRST DECIMAL DIGIT	POINTS PAST LAST DECIMAL DIGIT
PSW	IMMATERIAL	INDETERMINATE

COMDEC does not fail.

Table Manipulation Routines

The WIZARD virtual machine includes tables as a data type. Tables are constructed using strings and are single-keyed and unordered. A table is composed of zero or more table entries and is terminated by !EOR. While !EOR defines the end of table, the end of available table space may be indicated by a group-separator record. Routines are provided for inserting and deleting table entries, for looking up table entries based on their key values, for identifying and obtaining parameters within a table entry, and for generalized table housekeeping functions.

Table entries consist of fields separated by an augmented set of break characters (comma, space, tab, and the @). Fields within a table entry may contain any ASCII character except control characters (ØØH through 1FH) and three characters (! = @) that have a special significance within tables. Each table entry is bounded at beginning and end by an exclamation point. The end of a table (and the end of the last entry within that table) is indicated by !EOR. The items in the table entry are separated by space, tab, comma, and @. The @ sign is used before an address field. Addresses are coded as 4 ASCII hexadecimal digits, high-order to low-order, left to right. Parameter names are separated from their parameter values by an equal sign (=). A table entry may have only one key, but may have any number of parameters. A single table entry normally has a single address field; however, this convention is not enforced, and under certain circumstances, multiple address fields may be desirable.

!KEY,PAR1=VAL,PAR2=VAL@1000!

TBLFND--Locates a named entry in a single-keyed table. TBLFND is called with HL pointing to the address of the logical beginning of the table (the point from which the search begins). DE points to the first character of the key. The key must be terminated by an EOR character, a break character (space, comma, tab), or an exclamation point. On return, HL points to a location within the table entry associated with the key if the search was successful. If the search was unsuccessful, HL points to the address of the EOR character terminating the table. TBLFND fails if it is unable to find the key within the table.

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	POINTS TO KEY	INDETERMINATE
HL	POINTS TO START OF TABLE	POINTS TO TABLE ENTRY PARAMETER LIST OR EOR
A	IMMATERIAL	INDETERMINATE
FLAGS	IMMATERIAL	INDETERMINATE EXCEPT CARRY: CARRY SET AT Ø IF KEY IS IN TABLE; SET AT 1 IF NOT IN TABLE

FNDPAR--Finds a named parameter within a table entry. Before FNDPAR is called, TBLFND must be used to set HL. FNDPAR will not move HL to point past the current table entry. FNDPAR is called with DE pointing to the first character of the parameter name. The parameter name must be terminated by an EOR character. FNDPAR returns with HL pointing to the first character of the parameter value within the current table entry. DE is not preserved. If the named parameter does not occur within the current table entry, FNDPAR fails and HL is indeterminate except that it remains within the current table entry.

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	POINTS TO PARAMETER NAME	INDETERMINATE
HL	PRESET BY TBLFND	POINTS TO PARAMETER VALUE
A	IMMATERIAL	INDETERMINATE
FLAGS	IMMATERIAL	INDETERMINATE EXCEPT CARRY: CARRY SET AT Ø IF NAME PARAMETER IS FOUND; SET AT 1 IF NOT FOUND

GETADD--Obtains and decodes an address field from a table entry. GETADD finds the first address field in the current table entry and decodes it to a 16-bit binary address, which it returns in DE. The high-order byte is in D; the low-order byte is in E. Before GETADD is called, TBLFND must be used to leave HL pointing within the correct table entry. FNDPAR may be used subsequent to TBLFND, but prior to calling GETADD. GETADD will not move HL past the end of the current table entry. If the entry has no address field, or an illegal address field is detected (illegal hexadecimal numeric character--no range checking is performed), DE is set equal to Ø and GETADD fails.

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	IMMATERIAL	BINARY ADDRESS
HL	PRESET BY TBLFND	POINTS TO END OF ADDRESS FIELD
A	IMMATERIAL	INDETERMINATE
FLAGS	IMMATERIAL	INDETERMINATE EXCEPT CARRY: CARRY SET AT 1 IF GETADD FAILS; SET AT Ø IF IT SUCCEEDS

THISKE--Backs HL point to the first character of the key of the current table entry. HL must be pointing inside a table entry at the time of call. THISKE changes HL and PSW (the flags). THISKE does not fail.

NEXTKE--Moves HL forward to point to the first character of the key of the next table entry. HL must be pointing inside a table entry at the time of call. If HL points to the last entry of a table, NEXTKE fails and HL points to the table end-of-record. HL is changed; PSW is not preserved.

TBLBLD--Used to add a new entry to a table. TBLBLD is called with the address of the table in HL and the address of the new entry to be added to the table in DE. The new entry field pointed to by DE does not include an exclamation point and is terminated by an EOR character. TBLBLD assumes that the table has a group-separator record at the end of the available space. If there is an entry in the table with the

same key, the old entry is deleted. If there is inadequate room for the new entry, a garbage collection operation is performed in an effort to create adequate space. If room is still inadequate, TBLBLD fails. TBLBLD always adds the new entry at the bottom of the table. No registers are preserved.

TRASH--Places a garbage stamp on the beginning of the current table entry. TRASH assumes TBLFND has been used to set HL within a table entry. HL is left pointing within the invalidated table entry. PSW is not preserved. TRASH does not fail.

GARBAG--Collects garbage from the table pointed to by HL. On call, HL must be pointing at the ! at the beginning of a table. No registers are preserved. GARBAG fails if HL does not point to the ! on call. Note that since GARBAG moves strings to eliminate the space occupied by previously invalidated (TRASH) table entries, it may involve substantial processing overhead for large tables.

Augmented Arithmetic and Logic Instructions

These utility routines are used to augment the arithmetic and logic instruction set of the 8080. In general, they are used to provide 16-bit arithmetic and shift instructions.

DOUBLE REGISTER: SHIFT AND ROTATE SUBROUTINES--The Shift routine shifts the named register pair left/right one bit. The vacated low/high-order bit is filled by the CARRY bit. Other registers, including flags, are unchanged. The Rotate routine rotates the named register pair left/right one bit. The high/low bit which is shifted off the end is shifted around to fill the vacated low/high bit. Other registers, including the flags, are unchanged.

SHIFT		ROTATE	
SHLL:	HL left.	RHLL:	HL left.
SHLR:	HL right.	RHLR:	HL right.
SDEL:	DE left.	RDEL:	DE left.
SDER:	DE right.	RDER:	DE right.
SBCL:	BC left.	RBCL:	BC left.
SBCR:	BC right.	RBCR:	BC right.

SUBW--Subtract Word sets HL equal to HL minus DE (changes only HL and flags). Leaves flags indeterminate except CARRY as appropriate.

TESTW--Test Word in HL sets the flags based on the contents of HL interpreted as a 16-bit 2's-complement binary number. The accumulator is not preserved; other registers are unchanged.

Bit-Map Manipulation Routines

The WIZARD operating system incorporates a bit-map data type. A bit map is addressed by its zero byte. The zero and first bytes of a bit map give the number of data bytes in the map in address format (low, high). The 8 bits of each data byte of a bit map may be thought of as composing a linear array of logical variables. Individual bit set/reset/test routines accept 16-bit binary values as indices to access individual bits within a bit map. This means that the largest possible bit map consists of 64K individual bit entries occupying 8K bytes of storage. The number of bits in a bit map must be a multiple of 8.

SETMAP--Sets all bits in a bit map to 1. The zero and first bytes of the map give the number of bytes in the map in address format (low, high).

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	IMMATERIAL	INDETERMINATE
HL	POINTS TO MAP (Ø byte)	INDETERMINATE
PSW	IMMATERIAL	INDETERMINATE

SETMAP does not fail.

CLRMAP--Sets all bits in a bit map to Ø. The zero and first bytes of the map give the number of bytes in the map in address format (low, high).

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	IMMATERIAL	INDETERMINATE
HL	POINTS TO MAP (Ø BYTE)	INDETERMINATE
PSW	IMMATERIAL	INDETERMINATE

CLRMAP does not fail.

BITTST--Tests an individual bit in a bit map, returning the state of the bit as the CARRY flag. The zero and first bytes in a bit map are a count of the number of bytes in the map in address format (low, high). The succeeding bytes contain the bits to be tested. The 8 bits of each byte are ordered high to low, left to right. The data of bytes are ordered high to low, left to right. (Note: BITTST does not check to determine if the bit to be tested is outside the map.) On return, the CARRY flag is set equal to the state of the tested bit.

REGISTER	ON CALL	ON RETURN
BC	IMMATERIAL	UNCHANGED
DE	BIT NO. OF BIT TO BE TESTED	INDETERMINATE
HL	POINTS TO MAP (Ø BYTE)	INDETERMINATE
A	IMMATERIAL	INDETERMINATE
FLAGS	IMMATERIAL	INDETERMINATE EXCEPT CARRY: CARRY SET AT 1 IF TESTED BIT EQUALS 1; SET AT Ø IF TESTED BIT EQUALS Ø

BITTST fails if the tested bit is equal to 1.

COMMAND FORMAT

WIZARD commands consist of ASCII strings which are either entered at the system console or generated by user programs. In the description of each command the following conventions are used: Nonbracketed words are required and must be present exactly as shown. Lowercase nonbracketed words represent required command fields whose values are selected from the legal values for the specific commands. Words in angle brackets represent user-defined file or variable references. Expressions in square brackets are optional. Alternate forms are given above and/or below the preferred form. All command fields are separated by one or more break characters (space, tab, comma). An example is:

```
DIRECTORY [<FILREF>] [TO <OUTPUT DEVICE>]
DIR
```

WIZARD CONSOLE MONITOR COMMANDS

When engaged in direct interaction with the WIZARD console monitor, users direct the system to perform required tasks by typing console monitor commands. The system indicates its readiness to accept a console command by issuing a Carriage Return (CR) line feed and an exclamation-point prompt character. The user enters the command on the console keyboard and terminates it with a CR. Before CR is entered, typing errors may be corrected by using the delete key and/or control X. Each time the delete key is depressed, the last character in the string is deleted. Depending on the terminal handler, the deleted character may be erased and the string cursor backed up, or the deleted character may be echoed. Typing a control X prior to entering CR deletes the entire current command line. Typing a control R redisplays the current command line without altering any information.

Basic Monitor Commands

The basic monitor commands of WIZARD offer the user the facilities of a conventional memory resident microprocessor monitor.

GO--This command is used to transfer system control to a user program. Normally the user types the (hexadecimal) address of the starting point to his program as part of the command string.

```
G [<ADDR>]
G 4ØØØ
```

If Wizard gains control of the system through the execution of a breakpoint instruction or a WIZARD system call (RST 1), the transfer address may be omitted; in which case, the user program counter at the time of WIZARD access becomes the effective transfer address. If the WIZARD monitor was accessed through a breakpoint, then the breakpoint address

is the default. If a restart instruction was used as a WIZARD monitor call, the transfer address is to the instruction following the restart.

DISPLAY--There are two alternate forms of the display command: (1) The first and last addresses of a contiguous block of memory are explicitly given as command string arguments; (2) only the starting address is given explicitly, followed by the swath operator and the number of bytes of memory to be displayed. In all cases, memory is actually displayed in units of 16 bytes. All addresses and swath widths are entered in hexadecimal. The memory display output is directed to the console device and is provided as both a hexadecimal display of memory content and its ASCII equivalent.

```
D          <FIRST> <LAST>
D          <FIRST> S <SWATH>
D          4ØØØ  4ØFF
D          4ØØØ S 1ØØ
```

REGISTER DISPLAY--This command (R) is used to display user register contents.

SET REGISTER--This command is used to modify the contents of user registers.

```
SR   regl=<VALUE>,reg2=<VALUE>, . . . regn=<VALUE>
where regl . . . regn are selected from the values A,F,B,C,D,
  E,H,L,PC
SR   A=ØØ, H=4A, L=Ø6
```

When WIZARD is accessed either through encountering a breakpoint or by a WIZARD access call (RST 1), the user registers at the time of WIZARD access are saved. When control to the user program is restored by the command GO, user registers are restored. The Set Register command permits the user to modify the contents of these registers prior to returning control to the program. All registers displayed by the Display Register command can be modified, with the exception of the stack pointer. As many registers as desired can be modified with a single Set Register command; however, at least one register must be modified. Although any register can be set to hexadecimal value, the 8080 microprocessor restricts the value of certain bits of the flag register. If an attempt is made to set the flag register to an illegal value, the illegal bits are ignored.

SET MEMORY--This command is used to examine and modify the contents of selected memory address locations. The address field in the command string defines the address of the first byte to be examined and/or modified. After the Carriage Return is typed, terminating the command string, the system displays the current value of the selected byte.

```
S    <ADDR>
S    6ØØØ
```

The contents of this memory location can be modified by typing any legal two-digit hexadecimal number; after this, the succeeding memory location

is displayed. Typing a space causes the next memory location to be displayed without modifying memory contents. Memory display/modification is terminated by typing either CR or any nonhexadecimal digit except the space. During the data entry process, hexadecimal digits are entered immediately as the keys are struck, thus rendering the delete key ineffective.

INSERT MEMORY--This command is used to insert hexadecimal values into contiguous memory locations. The address field of the command string identifies the first memory location to be modified. After CR is typed, the system will display the address of the memory location to be inserted.

```
I    <ADDR>
I    6ØØØ
```

Any two-digit hexadecimal number may be entered, after which the system will display the next sequential address and await another value. Data entry is terminated by typing CR or other nonhexadecimal digit. Values are entered immediately as they are typed, thus rendering the delete key ineffective.

INPUT--This command is used to input a byte from any of the 256 input ports of the 8080. The input port is entered as a hexadecimal number in the range of Ø to FF. When the command is terminated with CR, the input byte is displayed.

```
IN   <PORT>
IN   BC
```

OUTPUT--This command is used to output a user-defined byte to any of the 256 8080-output ports. Both the output port and the byte to be output are entered as hexadecimal numbers in the range of Ø to FF.

```
OUT  <PORT> <VALUE>
OUT  BC   41
```

HEXLOAD--This command is used to load Intel hex-format paper tapes. Load address information is written on the hex file and is not included in the command string. If this command is used without the paper tape reader connected to the system, it will lock up the system, necessitating a hardware reset.

BREAKPOINT--This command offers the WIZARD user a powerful interactive debugging tool. The user can set up to four breakpoints in his program. When the processor attempts to execute an instruction at which a breakpoint has been set, normal processing is interrupted, the user registers are saved and displayed along with identification of the breakpoint encountered, and control is given to the WIZARD console monitor. The user may then examine and/or modify registers and memory locations, modify breakpoint locations, and return control to the user program either at the point it was interrupted or at some other location.

All Breakpoint commands begin with the symbol #. If the single symbol appears in isolation, it causes all previously set breakpoints to be cleared. When followed by a single digit 1 through 4, the specified breakpoint (1 through 4) is cleared; other breakpoints are unaffected. If the breakpoint identification digit is followed by a hexadecimal address, that breakpoint is set at the address given. If that breakpoint has previously been set at some other address, it is cleared before the new breakpoint is set.

```
#    [n[<ADDR>]]
#
#    1
#    4    6ØB3
```

The breakpoint facility utilizes 8080 restart instructions which replace the op code byte found at the breakpoint address. This places two restrictions on the use of breakpoints. First, breakpoints can be placed only in read/write/memory; they cannot be set in read-only memory. Breakpoints can be set only at instruction op code locations. A breakpoint located at an address or data field location within an instruction is invalid and may produce unpredictable system behavior. User program bugs may on occasion cause the execution of "ghost breakpoint." The most common example of this is when instructions that a program is attempting to execute have not been placed in memory. Most 8080 systems return a value of FF when the memory they attempt to read does not exist. This FF byte is interpreted as a restart instruction by the CPU simulating a breakpoint number 1. If this breakpoint has previously been set, the effect is identical to actually encountering the breakpoint. If the breakpoint has not been set or has been cleared, the breakpoint number 1 display will still appear; however, the program counter will be given as ØØØØ. This anomalous behavior is almost invariably caused by instruction execution at a nonexistent or noninitialized memory address.

CONSOLE DISK AND FILE MANAGEMENT

File Access Table

WIZARD's file access table (discussed in the IOCS Disk and File Management section) is used also in console disk and file management. The management commands DECLARE and ASSIGN can be used to modify the table.

Disk/File Management Commands

DECLARE--Used to associate a user-defined name with a disk file or physical device handler. This command is also used to delete files from the file access table.

```
                        NULL
DECLARE   <FILNAM>  FILE
                        DEVICE @<ADDR>
DECLARE   FILE1/Ø FILE
```

A file name is declared to the system as a disk file with the statement DECLARE <FILNAM> FILE. A file name may be associated with a physical device handler with the statement DECLARE <FILNAM> DEVICE @<ADDR>. In this case, the address is the hexadecimal value of the base address of the device handler. All device handlers specified in this manner must conform to the WIZARD device-handler specifications (Basic Input/Output Functions, p. 5). Any file name entered into the file access table by the DECLARE or ASSIGN statement may be declared NULL to delete it from the table. An attempt to nullify a nonexistent file results in an error message, but does not compromise system integrity. An already existing file name may be redeclared at any time, in which case the previous declaration is superseded. In all file names, the diskette drive designator (e.g. /Ø) is part of the file name.

ASSIGN--Used to associate a new file name with a previously defined file name.

```
ASSIGN    <FILNAM> TO <OTHERNAME>
ASSIGN    OUTPUT TO LO
```

In the example given, the logical file OUTPUT is associated with the predefined name LO (Listing Out). Subsequent references to the logical file OUTPUT are directed to LO. ASSIGN can be used to modify input/output assignments at run time.

CREATE--Interfaces to the PerSci intelligent diskette controller and is used to place file names into the diskette directory and to reserve storage for the files.

```
CREATE    <FILNAM>  <FILESIZE>
CREATE    FILE1/Ø   4Ø
```

The file name in the CREATE command string must be a legal disk file name (see File Reference section) and must not already exist on the diskette directory. The file-size field in the command string determines how many 128-byte diskette sectors are to be reserved for the file. It must be an integer value greater than Ø. The command CREATE does not utilize the file access table, and the name of the file to be created need not be declared prior to creation.

DELETE--Used to delete files.

```
DELETE    <FILNAM>
DELETE    FILE1/Ø
```

If an attempt is made to delete a nonexistent file, an error condition results; however, system integrity is not compromised. DELETE does not utilize the file access table, and a file name need not be declared prior to deletion. Deletion of a file does not remove that file name's file access table entry, if any.

LOAD--Used to load programs or other information from the diskette or other device.

```
LOAD      <FILNAM>
LOAD      MAIN/Ø
```

The name of the file to be loaded must be declared before LOAD is executed. A request to load a nonexistent file results in a system error; however, system integrity is not compromised. Upon completion of the load, program control is transferred to the loaded program or returned to the monitor, depending upon whether or not the transfer address was included in the termination record of the loaded file. (See SAVE command.)

LODFIL--Used to load programs from diskette files.

```
LODFIL    <FILNAM>
LODFIL    WAVR.5261Ø
```

This function first declares the file name to be a file, then invokes the LOAD command, and finally declares the file name to be null.

SAVE--Used to write programs or other information onto a diskette.

```
SAVE      <FILNAM>
SAVE      MAIN/Ø
```

The file name must be declared and the file created with a sufficient allocation before the command SAVE is executed. Information is saved as a block of binary information of arbitrary length. The SAVE routine requests entry of the first and last addresses of each block to be saved. These addresses are inclusive and are entered in hexadecimal. If desired, the entire 65K 8080 address space may be saved as a single block. To terminate a SAVE operation, the response END is entered instead of the block addresses. If the END string is followed immediately by a Carriage Return, then when the program is loaded, control is returned to the program requesting the load (or the console monitor, if the console command LOAD was used). If END is followed by a hexadecimal address before the CR, then when the program is loaded, control is passed to that address.

DIRECTORY--Used to obtain diskette directory information.

```
DIR
DIRECTORY [<FILREF>] [TO <OUTPUT DEVICE>]
DIR       */Ø TO LO
```

The command may be spelled out or abbreviated DIR. The default file reference is the user diskette (*/1). System diskette directory may be obtained using a file reference of */Ø. If directory information on only a specific file is desired, that file name should appear as the file reference. The default device for listing the directory is the console (predefined logical device CO) but may be changed by appending the TO OUTPUTFILNAM string to the DIRECTORY command string. In the example given, the directory of the system diskette would be output on the listing device. The command DIRECTORY causes a directory program to be loaded at AØØØ (hex) and executed.

COPY--Used to copy one disk file to another.

```
COPY      <OLDFIL> TO <NEWFIL>
COPY      */Ø TO */1
```

Under the current implementation, the command COPY can be used only for transferring information from one disk file to another. A new file is created and the information in the old file is copied to it. The original file remains unaltered. COPY does not utilize the file access tables, and neither the old file nor the new file need be declared. The new file must not already exist on the destination diskette or a system error will result; the already existing file will not be altered. In the example given, contents of the system diskette are transferred in toto to the user diskette. All diskette files must be closed before issuing the command COPY.

RENAME--Used to change the name of a diskette file or a group of diskette files.

```
RENAME    <OLNAM> TO <NEWNAM>
RENAME    MASTER  TO  OLDMASTR
```

The name of the file in the diskette directory is changed from the old name to the new name without modifying the contents of the file. In the example given, a disk file named MASTER is renamed OLDMASTR. The command RENAME does not utilize the file access table, and neither the old name nor the new name need be declared. The new file name must not be the same as any file already on the disk.

FORMAT--Used to initialize IBM soft-sectored format diskettes.

```
FORMAT    <DISKNAM>
FORMAT    SCRATCH
```

The argument to the command FORMAT is written onto the new diskette as the diskette name.

GAP--Used to compress the information on a diskette after files have been deleted. The command GAP accepts no arguments. All disk files must be closed prior to issuing the command GAP.

CLRDSK--After a user program error, CLRDSK (Clear Disk) is issued to insure that the PerSci intelligent disk controller has not left any files open. The command CLRDSK does not modify the WIZARD open-file table. A hardware reset will insure that the WIZARD file access table and open-file table are reset to their default values, and will automatically issue the command CLRDSK.